⑲日本国特許庁(JP)　⑪特許出願公開

⑫公開特許公報(A)　平2−242434

⑤1Int.Cl.[5]	識別記号	庁内整理番号
G 06 F 9/46	360 B	8945−5B
15/16	430	6745−5B

⑬公開　平成2年(1990)9月26日

審査請求　未請求　請求項の数 8　(全16頁)

⑤4発明の名称　タスクのスケジューリング方法

⑳特　願　平1−62216
㉒出　願　平1(1989)3月16日

⑦2発明者　上脇　正　茨城県日立市久慈町4026番地　株式会社日立製作所日立研究所内
⑦2発明者　山口伸一朗　茨城県日立市久慈町4026番地　株式会社日立製作所日立研究所内
⑦2発明者　斉藤雅彦　茨城県日立市久慈町4026番地　株式会社日立製作所日立研究所内
⑦2発明者　小林芳樹　茨城県日立市久慈町4026番地　株式会社日立製作所日立研究所内
⑦1出願人　株式会社日立製作所　東京都千代田区神田駿河台4丁目6番地
⑦4代理人　弁理士　秋本正実
最終頁に続く

明　細　書

1．発明の名称

タスクのスケジューリング方法

2．特許請求の範囲

1．全てが同一構成ではない複数のプロセッサと該プロセッサの各々によりアクセス可能な主メモリが共有バスで接続されて成るマルチプロセッサシステムでのタスクのスケジューリング方法において、各プロセッサに自プロセッサのスケジュール手段と自プロセッサの種別を示す識別子の認識手段とを有せしめ、主メモリに未処理タスクの待行列を格納し、かつ各タスクには自タスクを実行可能なプロセッサの種別を示す実行可否フラグを含むタスクテーブルを設けるとともに、いずれかのプロセッサにスケジューリングが必要となった場合に、当該プロセッサのスケジュール手段は、上記主メモリの待行列に繋がれたタスクを順次読み出し、自プロセッサの識別子と上記読み出したタスクの上記実行可否フラグとの比較から実行可能と判断した最初のタスクを実行タスクとすることを特徴とするタスクのスケージューリング方法。

2．全てが同一構成ではない複数のプロセッサと該プロセッサの各々によりアクセス可能な主メモリが共有バスで接続されて成るマルチプロセッサシステムでのタスクのスケジューリング方法において、いくつかのプロセッサにスケジュール手段を有せしめ、主メモリに未処理タスクの待行列と各プロセッサの種別を示す識別子表とを格納し、かつ各タスクには自タスクを実行可能なプロセッサの種別を示す実行可否フラグを含むタスクテーブルを設けるとともに、いずれかのプロセッサにスケジューリングが必要となった場合に、上記スケジュール手段を有するプロセッサの1つが、タスク処理中であればその処理を中断してスケジュール手段を起動し、上記主メモリの待行列に繋がれたタスクを順次読み出し、該読み出したタスクの上記実行可否フラグと上記スケジューリングが必要なプロセッサの上記識別子表から読み出した識別子との

比較から実行可能と判断した最初のタスクを実行タスクとすることを特徴とするタスクのスケジューリング方法。

3．全てが同一構成ではない複数のプロセッサと該プロセッサの各々によりアクセス可能な主メモリが共有バスで接続されて成るマルチプロセッサシステムでのタスクのスケジューリング方法において、各プロセッサに自プロセッサのスケジュール手段と自プロセッサの種別を示す識別子の認識手段とを有せしめ、主メモリに未処理タスクの待行列を格納し、かつ各タスクには自タスクを実行する上での各プロセッサ種別ごとの優先度を示す優先度テーブルを含むタスクテーブルを設けるとともに、いずれかのプロセッサにスケジューリングが必要となった場合に、当該プロセッサのスケジュール手段は、上記主メモリの待行列に繋がれたタスクを順次読み出し、自プロセッサの識別子に対して最も優先度の高いタスクを各タスクの上記優先度テーブルを参照して選び出し、該選び出したタスクを実

行タスクとすることを特徴とするタスクのスケジューリング方法。

4．全てが同一構成ではない複数のプロセッサと該プロセッサの各々によりアクセス可能な主メモリが共有バスで接続されて成るマルチプロセッサシステムでのタスクのスケジューリング方法において、いくつかのプロセッサにスケジュール手段を有せしめ、主メモリに未処理タスクの待行列と各プロセッサの種別を示す識別子表とを格納し、かつ各タスクには自タスクを実行する上での各プロセッサ種別ごとの優先度を示す優先度テーブルを含むタスクテーブルを設けるとともに、いずれかのプロセッサにスケジューリングが必要となった場合に、上記スケジュール手段を有するプロセッサの1つが、タスク処理中であればその処理を中断してスケジュール手段を起動し、上記主メモリの待行列に繋がれたタスクを順次読み出し、上記スケジューリングが必要なプロセッサの識別子に対して最も優先度の高いタスクを各タスクの上記優先度テ

ーブルを参照して選び出し、該選び出したタスクを実行タスクとすることを特徴とするタスクのスケジューリング方法。

5．前記優先度に実行不能を示す不能値を設け、あるタスクに対して上記不能値が指定された種別のプロセッサには当該タスクを割り当てないことを特徴とする請求項3または4記載のタスクのスケジューリング方法。

6．前記主メモリに代って各プロセッサ対応にローカルメモリを設け、前記共有バスに代って各プロセッサ間を結合するネットワークを設けるとともに、前記待行列はマスターとして指定された1つのプロセッサ対応のローカルメモリに格納し、上記待行列への他プロセッサからのアクセスを上記ネットワークを介して行うことを特徴とする請求項1，2，3，4または5記載のタスクのスケジューリング方法。

7．前記主メモリに代って各プロセッサ対応にローカルメモリを設け、前記共有バスに代って各プロセッサ間を結合するネットワークを設けるとともに、前記待行列はすべてのプロセッサ対応のローカルメモリに格納し、待行列の更新時には上記ネットワークを介してすべてのローカルメモリで行うようにしたことを特徴とする請求項1，2，3，4または5記載のタスクのスケジューリング方法。

8．前記主メモリに代って各プロセッサ対応にローカルメモリを設け、前記共有バスに代って各プロセッサ間を結合するネットワークを設けるとともに、すべてのローカルメモリの物理空間アドレスを論理空間のアドレスに対応づけるためのアドレス変換手段を各プロセッサに設けることにより単一の仮想メモリを形成し、該仮想メモリを各プロセッサの共有メモリとしたことを特徴とする請求項1，2，3，4または5記載のタスクのスケジューリング方法。

3．発明の詳細な説明

〔産業上の利用分野〕

本発明は、マルチプロセッサにおけるタスクのスケジューリング方法に係わり、特に構成の異な

るプロセッサを含むマルチプロセッサにおけるタスクのスケジューリング方法に関する。

〔従来の技術〕

従来マルチプロセッサでは、これを構成する複数のプロセッサが、全く同じ構成を、従って同じ命令セットを持つことを条件としたものが多かった。

また構成が異なり、従って命令セットの異なるプロセッサを接続したマルチプロセッサやマルチコンピュータの場合には、処理すべきタスクが入る待行列がプロセッサごとに設けられており、タスクの実行に先立ってユーザまたはＯＳがそのタスクをどのプロセッサの上で実行するか決定し、その決定したプロセッサの待行列にタスクを入れておく必要があった。

また、特開昭62－208157には、スケジューリング方法ではないが、一部命令セットが異なる複数の計算機からなる計算機システムのプログラム実行方法が記載されている。その方法は、コンパイラがどの計算機の上ででも実行可能なオブジェクトコードを出力するものである。即ちコンパイラはプログラムを解析して、プロセッサ間で共通の命令で実行できる部分と実行できない部分に分ける。共通に実行できる部分については単一のオブジェクトコードを出力するが、実行できない部分はプロセッサの種類に対応した複数のオブジェクトコードを出力する。そして、コンパイラは、実行できない部分の直前に実行しているプロセッサの種類に応じたオブジェクトコードに分岐するためのプログラムを挿入する。

〔発明が解決しようとする課題〕

上記した従来のマルチプロセッサにおけるタスクのスケジューリング方法あるいはマルチプロセッサ用のプログラムコンパイル方式には以下のような問題がある。

即ち、同一構成のプロセッサのみでマルチプロセッサを構成する場合には、ベクトルプロセッサを備えたプロセッサやＡＩ処理用プロセッサなどの一つの機能を強化したプロセッサをシステム中に加えるときに、一つだけのプロセッサを入替えることができず、全部のプロセッサをそれらのプロセッサにしなければならない。

プロセッサ毎に待行列を持つ方法では、同じ構成のプロセッサが複数あるときにプロセッサ間で負荷が偏る可能性がある。

コンパイラが複数のオブジェクトコードを出力する方法では、オブジェクトコードが大きくなる欠点とともに、コンパイラを新しく作り直す必要があるという欠点がある。

本発明の目的は、複数のオブジェクトコードを出力するコンパイラを作成することなしに、命令セットの一部または全部が異なるプロセッサよりなるマルチプロセッサ上で、タスクにプロセッサを効率よく割付けるタスクのスケジューリング方法を提供することにある。

本発明の別の目的は、ベクトルプロセッサを持ったプロセッサと持たないプロセッサのように、ある処理が速いプロセッサと遅いプロセッサとを含むマルチプロセッサにおいて、そのような処理があるタスクはその処理の速いプロセッサに優先的に割当てることのできるタスクのスケジューリング方法を提供することにある。

また、本発明の別の目的は、ある処理が速いプロセッサにその処理を含むタスクを優先的に割当てるとともに、ある処理が不可能なプロセッサにはその処理を含むタスクを割当てないようにするタスクのスケジューリング方法を提供するにある。

〔課題を解決するための手段〕

上記の目的は、各タスクにそのタスクを実行できるプロセッサの種類を示すフラグを設けることにより達成され、各タスクに、そのタスクを処理できる速度に応じた優先度をプロセッサの種類ごとに示す優先度テーブルを設けることにより達成され、また上記優先度に不能値を設けこの値が指定された種類のプロセッサには当該タスクの処理が不可能とすることにより達成される。

〔作　用〕

各タスクに設けるフラグは、システムに含まれるプロセッサの種類の数だけのフラグ(１ビット)から成る。各フラグは、各プロセッサの種類に対

応しており、そのタスクがその種類のプロセッサで実行可能かどうかを表す。実行可能な状態にあるタスクは、どのプロセッサからもアクセス可能な待行列に繋がれている。アイドルとなったプロセッサやリスケジューリングを行うプロセッサは、各自、その待行列の先頭からタスクのフラグを順番に見ていき、自分のプロセッサで実行可能なタスクがあったら、取り出してきて実行する。これにより、同一種類のプロセッサ間では負荷の片よりが防がれる。

また、各タスクに設ける優先度情報テーブルは、システムに含まれるプロセッサの種類の数の大きさの整数配列とする。各配列要素は、各プロセッサの種類に対応しており、そのタスクがその種類のプロセッサに対して持つ優先度を示し、そのタスクを高速に実行できるプロセッサについては、高い優先度を入れておく。優先度を最低にしたときはその種類のプロセッサでは、そのタスクが実行不可能であることを表すとする。以上のようなテーブルを備えた各タスクは、実行可能な状態に

あるとき、どのプロセッサからもアクセス可能な待行列に繋がれている。アイドルとなったプロセッサやスケジューリングを行うプロセッサは、各自、待行列のタスクを見ていき、自分のプロセッサで実行可能で、優先度が最も高いタスクを取り出してきて実行する。

この方法により、命令セットが違うプロセッサやある処理が特別に速いプロセッサを含むマルチプロセッサで効率良く、タスクにプロセッサを割振ることができる。

〔実施例〕

以下、本発明の実施例について図面を参照して説明する。第１図は本発明の一実施例を示すもので、プロセッサ1011～1013は、全て共有バス103を通してメインメモリ104にアクセス可能である。このうちプロセッサ1011，1012は同一の命令セットを持つタイプ０のプロセッサ、プロセッサ1013は別の命令セットを持つタイプ１のプロセッサとする。このプロセッサのタイプはプロセッサ種類識別子1021～1023で示されている。

メインメモリ104には未処理タスク1061～1063の待行列が記憶されている。この待行列は各タスク同志を各タスクがもつポインタ1561，1562で繋ぐことにより形成されている。待行列の先頭はヘッダ105である。タスク1064～1066は、現在プロセッサ1011～1013で実行されている。待行列中のタスク1061～1063には実行可否フラグ1071～1073が設けられており、各ビット位置のフラグがプロセッサ種類に対応している。そしてフラグが“１”のときは対応する種類のプロセッサでそのタスクを実行でき、“０”のときは実行できない。例えば実行可否フラグ1071は、タスク1061をタイプ２とタイプ１のプロセッサでは実行できるが、タイプ０のプロセッサでは実行できないことを示している。

第２図は、プロセッサ種類識別子1021～1023の内容を示しており、３ビットより成っていて１ビットのみが“１”になっている。このプロセッサ種類識別子1021～1023は、実行可否フラグ1071～1073との論理的な積和をとることにより、その種

類のプロセッサでそのタスクを実行できるか否かを調べられるように構成されている。

第３図はスケジューラプログラム（以下では単にスケジューラとも呼ぶ）のフローチャートを示す。このスケジューラは、各プロセッサ1011～1013に設けられており、当該プロセッサが実行していたタスクが終了したときと、スケジュールのためのタイマ割込みが入ったときに起動される。タイマ割込みは、各プロセッサに一定時間間隔で入る。今プロセッサ1011にタイマ割込みが入って、スケジューラが起動されたとすると、まずステップ201で自プロセッサの種類を示す識別子1021を読み込む。読み込んだ値は“１００”である。次にステップ202で待行列のヘッダ105の指すタスク1061を見る。ステップ203では待行列にタスクがないか調べ、未処理のタスクがなかったときはアイドル処理のステップ204へ移る。第１図ではタスク1061が有るからステップ205でタスクの実行可否フラグとプロセッサ種類識別子の論理的な積和Ｓを求める。これは両者の各ビットごとのアン

ドをとり、それらの結果をオアしたもので、どこかのビット同志のアンドが“１”になればそれは当該タスクを当該プロセッサで実行できることを示す。今の場合、実行可否フラグ1071は“１１０”であり、プロセッサ種類識別子1021は“００１”だったので、各ビットごとのアンドは全て“０”、従ってＳ＝０で、プロセッサ1011ではタスク1061は実行できないことがわかる。そこでステップ207へ進み、待行列の次のタスクを見る。続くステップ203でタスクがないか調べ、タスク1062が見つかるとステップ205で実行可否フラグ1072とプロセッサ種類識別子1021の論理的積和Ｓを求める。今度は“００１”と“００１”なので各ビットのアンドは“０”,“０”,“１”となり、そのオアは“１”つまり実行可能なことがわかる。それでステップ208でタスク1062を待行列から取り除く。このときプロセッサ1011で実行中であったタスク（タイマ割込みで中断された）はステップ209で待行列の後端に接続される。これら待行列のタスクの出，入はポインタ変更により行われる。

そしてステップ209で待行列から取り除いたタスク1062を実行する。

第４図は、各プロセッサによるタスクの実行例をタイムチャートで示したものである。同図ではプロセッサ1011～1013による処理のタイムチャートがＰ0～Ｐ2で示されており、タイマ割込み301～309およびタスクの終了350によりスケジューラプログラムが動作し、タスク360～370が各プロセッサ上で実行されている様子を示している。即ち、最初はプロセッサ1011～1013は、それぞれタスクＴ3，Ｔ4，Ｔ5を実行していたとする。プロセッサ1011にタイマ割込み301が入ると、第３図で説明したようにこのプロセッサにはタスクＴ1がスケジュールされる。次にプロセッサ1012にタイマ割込み304が入ってタスクＴ2が実行され、プロセッサ1013にタイマ割込み307が入り、タスクＴ0が実行される。プロセッサ1012では、実行されていたタスクＴ2の終了350により、タスクＴ3が再び実行される。以下、同様に処理が続く。

第５図は、第１図のタスク1061～1066を実現す

るためのタスクテーブルを表す。タスク識別子150はタスクに付けたシリアル番号であり、信号をタスクに送るときなどにタスクを識別するのに使われる。タスクの状態151はタスクが現在どのような状態にあるかを表しており、以下のようなものがある。

User Running‥‥プロセッサ上でタスクのユーザプログラムを実行している状態。

Kernel Running‥‥プロセッサ上でユーザプログラム実行中にシステムコールを行ったり、割込みが入ったりしたためシステムプログラムを実行している状態。

Ready to Run‥‥タスクは実行できる状態になっているが、まだ、プロセッサが割当てられていない。タスクは待行列に繋がれている。

Sleep‥‥‥Ｉ／Ｏの応答待ちなど、ある事象が起るのを待っている状態。

優先度152はタスクが実行される優先度を表す数値で、この数値が大きいほど先にプロセッサで実行される。プロセッサが割り当てられるのを待

っているタスクの待行列は、この優先度が大きい順に並べられる。

実行可否フラグ107は、第１図のフラグ1071～1073と同じものであり、本発明の実現のために設けられた。これは当該タスクがどの種類のプロセッサで実行できるかの情報が入る。シグナル・フィールド153は、タスク間でタスクの中断，終了，エラーなどの情報を送るシグナルが入るフィールドである。タイマー108は、タスクの実行時間、システムの実行時間などの時間の情報が入るフィールドである。リージョン・テーブル154は、タスクで使われるプログラムやデータが記憶されているリージョンのページテーブルを指すポインタとリージョンの大きさの情報が入っている。レジスタ退避領域155には、タスクがタイマー割込みにより中断されたときなどに、中断される前のタスクのレジスタの情報を退避する領域である。中断されたタスクがスケジュールにより再開されるときには、ここに退避してあったレジスタの情報をもとにタスクを再開する。ポインタ156，157は

第１図で述べたタスクを待行列に繋ぐときのポインタである。

以上に説明した第１図の実施例では、スケジューラプログラムは各プロセッサに設けられており、各プロセッサは自分のスケジューリングを自分で行うものとしたが、これを負荷の最も軽い状態にあるプロセッサでスケジュールするようにすることもできる。第６図はそのようにした実施例を示している。これは第１図の構成と殆ど同じであるが、異なるのはシステム中のプロセッサの種類を記憶した表110をメインメモリに記憶している点である。表110へは、システム立ち上げ時に各プロセッサが自分のプロセッサ種類識別子1021～1023を読み出して表110に書き込み、これによってスケジューラ109は今スケジュールしようとしているプロセッサの種類を識別する。

第７図はこの実施例におけるスケジューラプログラムのフローチャートであり、スケジュールのためのタイマ割込みが入ったときにいずれか負荷の低いプロセッサで実行される。第６図の状態、

即ちプロセッサ1012にタイマ割込みが入って、スケジューラプログラムが実行されたとすると、まずステップ701で待行列のヘッダ105の指すタスク1061を見る。ステップ702で、この見たところにタスクがないか調べる。未処理のタスクがなかったときは、スケジューラプログラムは何もせずに終了するが、今はタスク1061が有ったのでステップ703へ進む。ステップ703では、タスクの実行可否フラグで実行可能となっているプロセッサの内(これをメインメモリ104の表110を用いて調べる)、アイドルのプロセッサがあればそのプロセッサを選び、なければ現在処理しているタスクの処理時間を第５図のタイマ108より調べ、それが最も長いプロセッサを選ぶ。第６図では実行可否フラグが“１１０”なので実行可能なプロセッサの種類はタイプ１とタイプ２である。これに該当するのはプロセッサ1013しかないのでこのプロセッサがここでは選択される。ステップ704では選択したプロセッサが実行していたタスクを中断し、その状態をセーブする。選択されたプロセッサ1013で

はタスクＴ5が実行されていたから、このタスクを中断し、中断したときのレジスタの内容をタスクＴ5のタスクテーブル内のレジスタ退避領域155に退避する。ステップ705では、中断したタスクＴ5を待行列に入れる。入れる場所はタスクテーブルに書かれている優先順位152が大きい順に並ぶように挿入する。最後にステップ706で選択したタスク1061（Ｔ6）を待行列より取り出して、選択したプロセッサ1013で実行させる。

第８図は、各プロセッサによるタスクの実行例を示すタイムチャートで示したものである。プロセッサ1011～1013による処理のタイムチャートがＰ0～Ｐ2で示されており、スケジューラプログラムＳの終了330～332の時にコンテキストスイッチ340～342によって実行タスクが切り換えられる。最初、プロセッサ1011～1013はそれぞれタスクＴ3、スケジューラプログラムＳ、タスクＴ5をそれぞれ実行していたとする。第７図を用いて説明したように、スケジューラＳはプロセッサ1013で実行していたタスクＴ5をコンテキストスイッチ

342により中断させ、タスクＴ0を実行させる。スケジューラＳが終了するとプロセッサ1012は以前に実行していたタスクＴ4の実行に戻る。一定時間が過ぎると再びタイマ割込み310が入り、スケジューラＳが実行される。このスケジューラは、第６図のタスクＴ1を待行列より取り出してくる。タスクＴ1の実行可否フラグ1072は“００１”なのでタイプ０のプロセッサで実行できる。これはプロセッサ1011，1012であるが、どちらのプロセッサもタスクを実行中なので現在実行しているタスクの実行時間の長い方のプロセッサを選択する。第８図ではプロセッサ1011の方が長かったとして、このプロセッサ1011でタスクＴ1を実行させている。以下同様に一定時間間隔でタイマ割込みにより、いずれかのプロセッサでスケジューラが実行され、待行列のタスクがスケジュールされていく。

第９図は本発明の別の実施例を示す図である。本実施例では各プロセッサ1011～1013は、その識別子1021～1023に示されているように、構成が異なっており、命令セットが一部異なっている。メ

インメモリ104内の待行列に繋がれたタスク1061～1063の各々には、各種類のプロセッサがタスクに対してどのような優先度を持つかを記憶したテーブル1081～1083が設けられている。テーブル1081の最初の行は、プロセッサ種類識別子001を持つタイプ0のプロセッサがタスク1061に対して0の優先度を持っていること、すなわち、タイプ0のプロセッサは、このタスクを実行できないことを表している。以下の行はタイプ1のプロセッサが10の優先度をもち、タイプ2のプロセッサは20の優先度を持つことを表す。

第10図は第9図の実施例で使用するスケジューラプログラムのフローチャートである。このスケジュールプログラムは第3図のスケジューラプログラムと同様に、実行していたタスクが終了したときと、スケジュールのためのタイマー割込みが入ったときに実行される。タイマー割込みは、各プロセッサに一定時間間隔で入る。第9図の状態でプロセッサ1013にタイマー割込みが入って、スケジューラプログラムが実行される場合を説明する。まず、ステップ1001で自分のプロセッサのプロセッサ種類識別子1021を読込む。読込んだ値は010である。次にステップ1002で待行列のヘッダ105にタスクが接続されているか調べている。タスクが接続されていない場合は、処理すべきタスクが無いことを示しているのでアイドル処理1003を行う。ここではタスク1061が接続されているのでYesの方に進み、ステップ1004で待行列に接続されているタスクのうち、プロセッサ種類識別子に対応した優先度が最大のタスクを選択する。第9図ではタスク1061～1063が待行列に接続されており、プロセッサ種類識別子010に対応した優先度は、それぞれ10，0，20である。よって、優先度20を持つタスク1063が選択される。選択されたタスクの優先度は0ではないので、ステップ1005からステップ1006へ進む（もし、選択されたタスクの優先度が0だった場合は、自プロセッサの種類では実行できるタスクがないということなので、アイドル処理1003を行う）。ステップ1006ではタスク1063を待行列から取り除く。即ちポインタ

1562をなくする。最後にステップ1007でタスク1063を実行する。

第11図は、第9図の各タスクを実現するためのタスクテーブル106を示す。第5図のテーブルと比較して異なるのは、第5図の実行可否フラグ107がなくなり、第5図の優先度152がプロセッサ種類ごとの優先度158になった以外は同じである。実行可否フラグで表現していたプロセッサの種類ごとによるタスク実行の可否は、優先度158が0か否かで表現している。

第12図は本発明の別の実施例を示す図である。第9図の実施例では、各プロセッサがスケジューラプログラムを持ち、自プロセッサのスケジューリングを行うとしたが、本実施例ではプロセッサの種類を記憶した表110をメインメモリ104に記憶しておき、これを用いていずれかのプロセッサがそのとき必要なスケジューリングを行う。このため、システム立ち上げ時に各プロセッサが自分のプロセッサ種類識別子1021～1023を読み出して表110に書き込んでおく。

第13図は本実施例におけるスケジューラのフローチャートを示す。スケジューラプログラムは、スケジュールのためのタイマー割込みが入ったときにいずれか負荷の低いプロセッサで実行される。そこで、第12図のように、プロセッサ1011にタイマー割込みが入って、ここでスケジューラプログラム109が実行された場合を説明する。まず、ステップ1301で待行列のヘッダ105の指すタスク1061を見る。ステップ1302でタスクがないか調べ、未処理のタスクがなかったときは、スケジューラプログラムはなにもせずに終了する。ここでは、タスク1061が有ったのでステップ1303へ進み、タスク1061の優先度テーブル1081を見て、優先度が最も高くなっているプロセッサ種類を選択する。ここでは優先度20をもつプロセッサ種類識別子100（タイプ2）のプロセッサが選択される。ステップ1304では選択されたプロセッサ種類のプロセッサを表110より調べ、プロセッサ1012がその種類のプロセッサであることがわかる。ここではその種類のプロセッサが1つしかなかったが、複

数あったときは現在そのプロセッサが実行しているタスクの実行時間が最も長いプロセッサを選択する。ステップ1305では選択したプロセッサが実行していたタスクを中断し、その状態をセーブする。選択されたプロセッサ1012ではタスクT4が実行されているから、このタスクを中断し、中断したときのレジスタの内容をタスク1065（T4）のタスクテーブルのレジスタ退避領域155（第11図）に退避する。ステップ1306では、中断したタスクT4を待行列の最後に入れる。最後にステップ1307で選択したタスク1061を待行列より取り出して、選択したプロセッサ1012で実行させる。

第14図は本発明の別の実施例を示す図である。本実施例では3個のプロセッサ1011～1013がネットワーク300に接続され、各々はローカルメモリ3101～3103を有している。第1図，第6図，第9図，第12図の実施例では、各プロセッサ全てからアクセス可能なメインメモリ104に待行列を記憶させていたが、本実施例では待行列はマスターとなるプロセッサ1011のローカルメモリ3101に記憶

されている。このプロセッサ1011以外のプロセッサがスケジューリングのために待行列にアクセスする必要があるときは、ネットワーク300とプロセッサ1011を通してローカルメモリ3101にアクセスする。実行可能か否か、および優先度に基づくスケジューリングの方法は既に述べた実施例と同じである。

第15図は本発明の別の実施例を示す図である。本実施例は、第14図の実施例の変形例であり、マスターとなるプロセッサがなく、すべてのローカルメモリに同一の待行列が記憶されている。即ち各ローカルメモリ3101～3103のヘッダ1051～1053には、同一のタスク1061，1062が繋がれており、各プロセッサは、自分のローカルメモリの待行列を見てスケジューリングする。待行列の更新は、他の全てのプロセッサに許可を得てから、自分のローカルメモリ内で行うとともに、各自のローカルメモリでも更新するよう、他プロセッサに指示するメッセージをネットワーク300に放送する。

第16図は、本発明の別の実施例を示す図であり、

第15図の実施例の各ローカルメモリ3101～3103を1つの仮想共有メモリ320としてアクセスする構成としたものである。どのプロセッサからでも任意のローカルメモリへアクセスでき、スケジューリングに用いる待行列もこの仮想共有メモリに記憶させる。

第17図は仮想共有メモリ320の論理アドレスと物理アドレスのマッピングの例を示す。仮想共有メモリの論理アドレス空間500の各アドレス領域がローカルメモリ3101～3103の物理アドレス空間5101～5103に対応づけられる。この対応づけを行うためのアドレス変換テーブルが第8図に示されており、例えばあるプロセッサが論理アドレス0をもつページをアクセスしたとする。このページのプロセッサ番号ID0が自プロセッサの番号であれば、自分のローカルメモリ内の物理アドレス0のページより読み出す。プロセッサ番号が異なる場合は、アクセスしたプロセッサは、番号ID0をもつプロセッサに物理アドレス0のページを転送するように、ネットワーク300を通してメッセージ

を送る。メッセージを受けたプロセッサは要求されたページをメッセージにして要求元に送る。この実施例においても、タスクごとの実行可否あるいは優先度にもとづく制御は先の実施例と同様である。

〔発明の効果〕

本発明によれば、アーキテクチャが異なり、命令セットも一部異なるプロセッサよりなるマルチプロセッサがあるとき、ユーザは、タスクをどのプロセッサで実行させるかを意識する必要がないのでタスクの管理効率が大幅に向上する。

また、本発明によれば、命令セットは共通であるが、ベクトルプロセッサなどにより、ある種の処理が強化してあるプロセッサを含むマルチプロセッサで、タスクの処理内容により最適なプロセッサを割当てるスケジューリングが提供されるのでシステムのスループットが向上する。

4．図面の簡単な説明

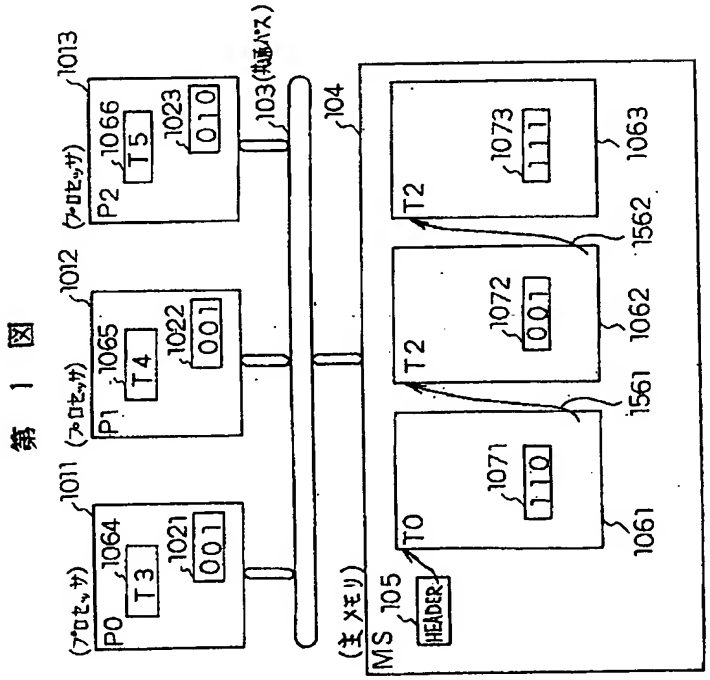
第1図は本発明の一実施例を示す図，第2図はシステムを構成するプロセッサの種類とプロセッ

サ種類識別子の対応を示す図、第3図は第1図の実施例で用いるスケジューラのフローチャート、第4図は第1図の実施例の動作例を示すタイムチャート、第5図は第1図の実施例におけるタスクテーブルの構成例を示す図、第6図，第7図および第8図は本発明の別の実施例を示す図、用いるスケジューラのフローチャート、および動作例を示すタイムチャート、第9図，第10図および第11図は本発明の別の実施例を示す図、用いるスケジューラのフローチャート、およびタスクテーブルの構成例を示す図、第12図および第13図は本発明の別の実施例で示す図および用いるスケジューラのフローチャート、第14図及び第15図はそれぞれ各プロセッサがローカルメモリを有してネットワークで結合されたシステムでの本発明の実施例を示す図、第16図は各プロセッサが有するローカルメモリを1つの仮想メモリとしてアクセスするようにしたシステムでの本発明の実施例を示す図、第17図および第18図は第16図の実施例における論理空間と物理空間の対応づけの説明図および上記2つの空間の対応づけを行うアドレス変換テーブルの説明図である。

1011～1013…プロセッサ、1021～1023…プロセッサ種類識別子、103…共有バス、104…メインメモリ、105…ヘッダ、1061～1066…タスク、1071～1073…実行可否フラグ、1081～1083…優先度テーブル、109…スケジューラプログラム、110…プロセッサ種類識別子表、3101～3103…ローカルメモリ、320…仮想共有メモリ。

代理人弁理士　秋本　正実

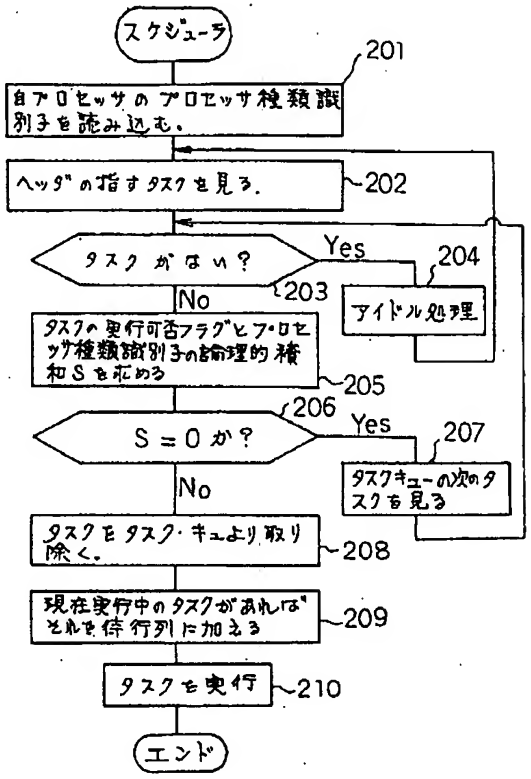
第1図



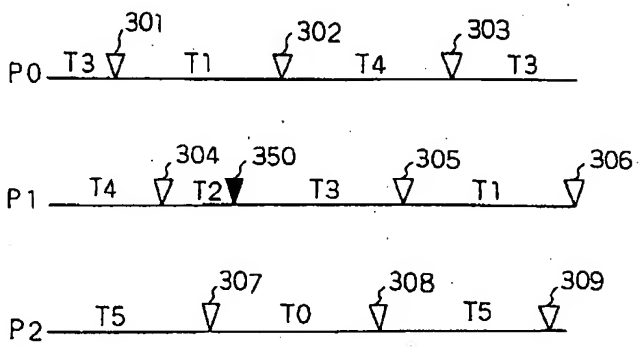
第2図

プロセッサ種類	プロセッサ種類識別子
タイプ 0	001
タイプ 1	010
タイプ 2	100

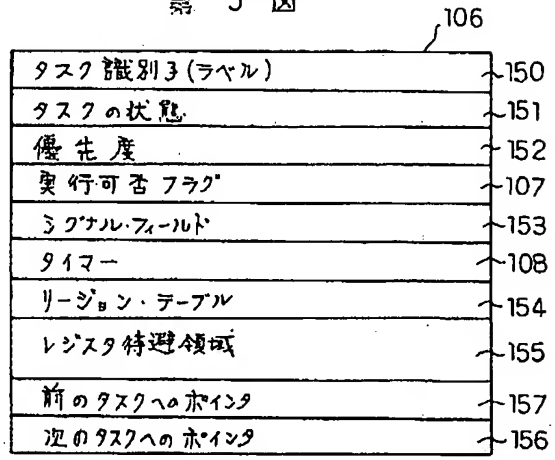
第 3 図


スケジューラ
201
自プロセッサのプロセッサ種類識別子を読み込む。
ヘッダの指すタスクを見る。
202
タスクがない？
Yes
204
No
203
アイドル処理
タスクの実行可否フラグとプロセッサ種類識別子の論理的積和Sを求める
205
206
S＝0か？
Yes
207
No
タスクキューの次のタスクを見る
タスクをタスク・キューより取り除く。
208
現在実行中のタスクがあればそれを待行列に加える
209
タスクを実行
210
エンド

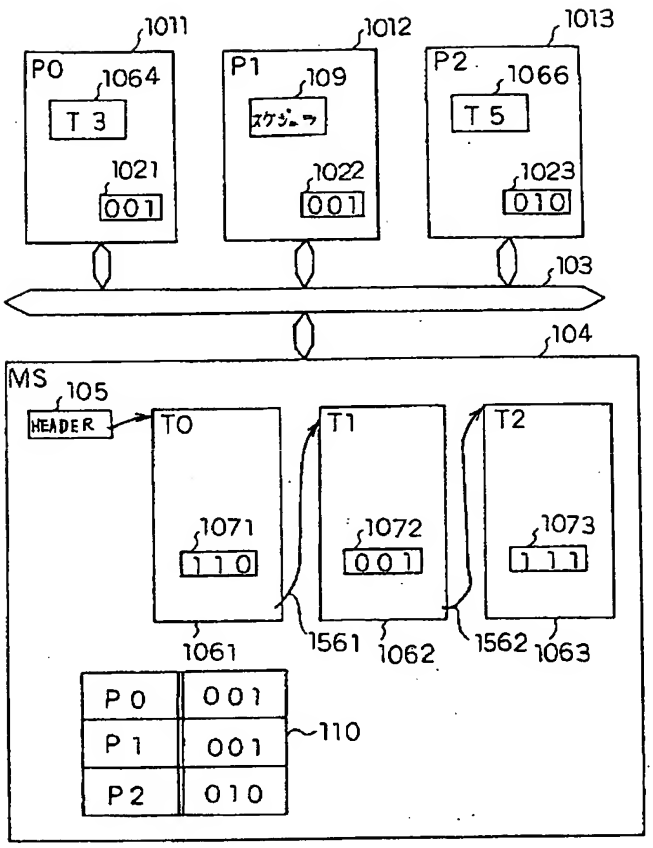
第 4 図


301
302
303
P0
T3
T1
T4
T3
304
350
305
306
P1
T4
T2
T3
T1
307
308
309
P2
T5
T0
T5

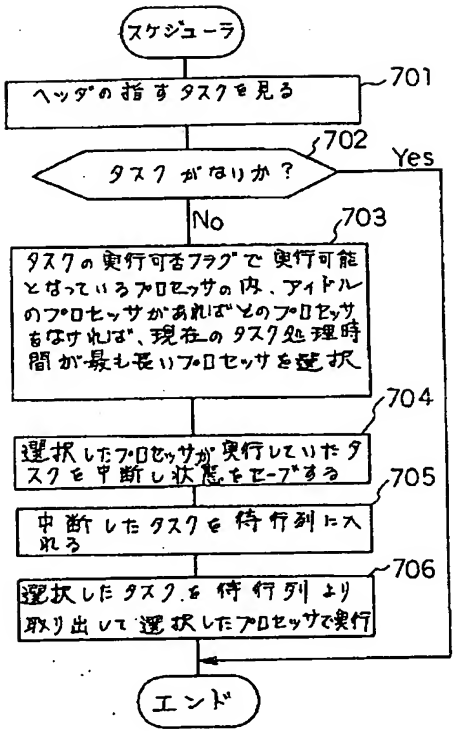
第 5 図


106
タスク識別子（ラベル）
150
タスクの状態
151
優先度
152
実行可否フラグ
107
シグナル・フィールド
153
タイマー
108
リージョン・テーブル
154
レジスタ待避領域
155
前のタスクへのポインタ
157
次のタスクへのポインタ
156

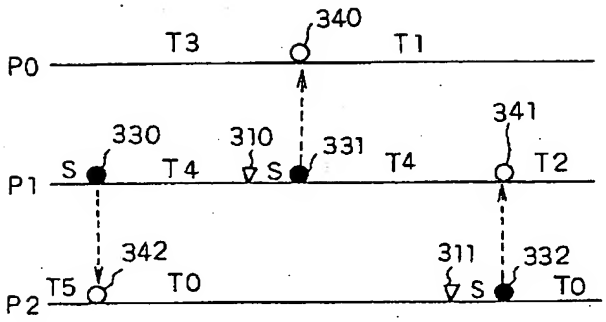
第 6 図


1011
1012
1013
P0
1064
T 3
1021
001
P1
109
スケジューラ
1022
001
P2
1066
T 5
1023
010
103
104
MS
105
HEADER
T0
T1
T2
1071
110
1072
001
1073
111
1061
1561
1062
1562
1063
P 0
001
P 1
001
P2
010
110

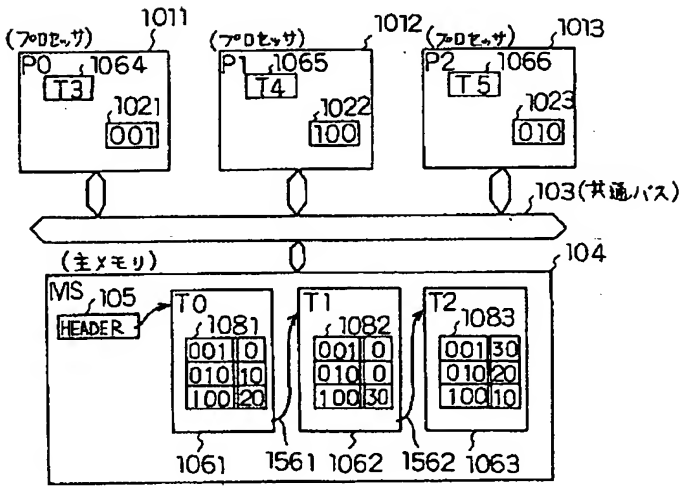
第 7 図


スケジューラ
ヘッダの指すタスクを見る
701
タスクがないか？
702
Yes
No
703
タスクの実行可否フラグで実行可能となっているプロセッサの内、アイドルのプロセッサがあればそのプロセッサをなければ、現在のタスク処理時間が最も長いプロセッサを選択
704
選択したプロセッサが実行していたタスクを中断し状態をセーブする
705
中断したタスクを待行列に入れる
706
選択したタスクを待行列より取り出して選択したプロセッサで実行
エンド

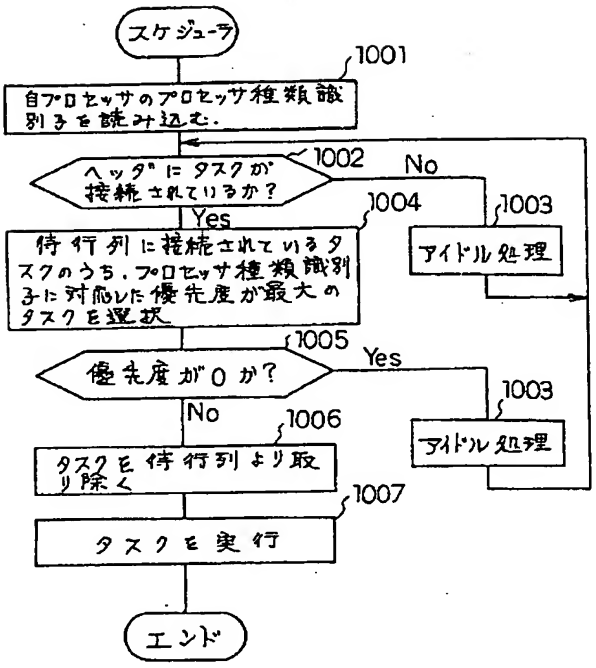
第 8 図


340
T3
T1
P0
341
330
310
331
S
T4
S
T4
T2
P1
342
311
332
T5
T0
S
T0
P2

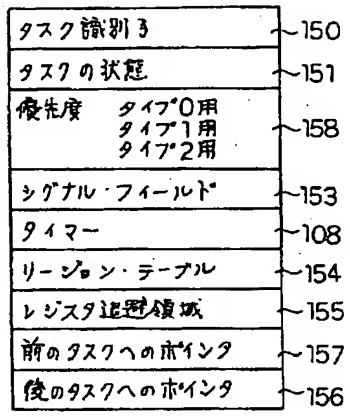
第 9 図


1011
(プロセッサ)
P0
1064
T3
1021
001
1012
(プロセッサ)
P1
1065
T4
1022
100
1013
(プロセッサ)
P2
1066
T5
1023
010
103(共通バス)
104
(主メモリ)
MS
105
HEADER
T0
1081
001 0
010 10
100 20
T1
1082
001 0
010 0
100 30
T2
1083
001 30
010 20
100 10
1061
1561
1062
1562
1063

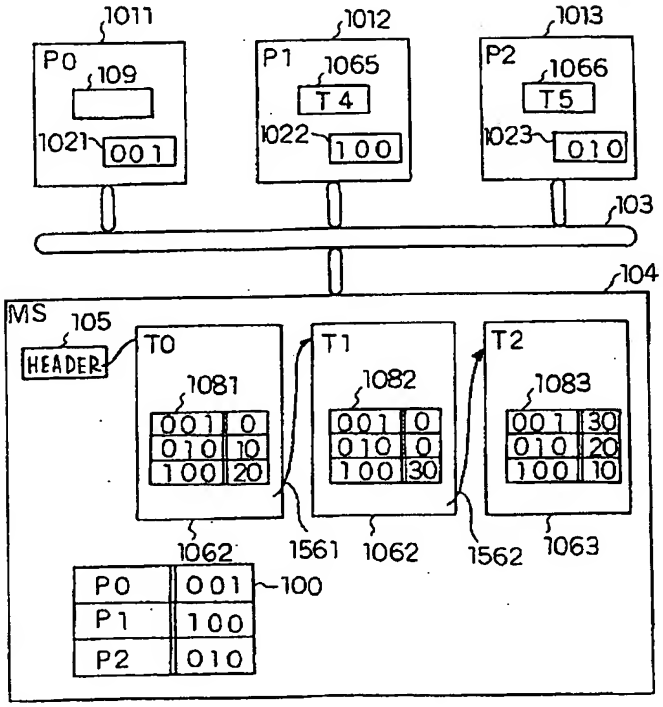
第 10 図



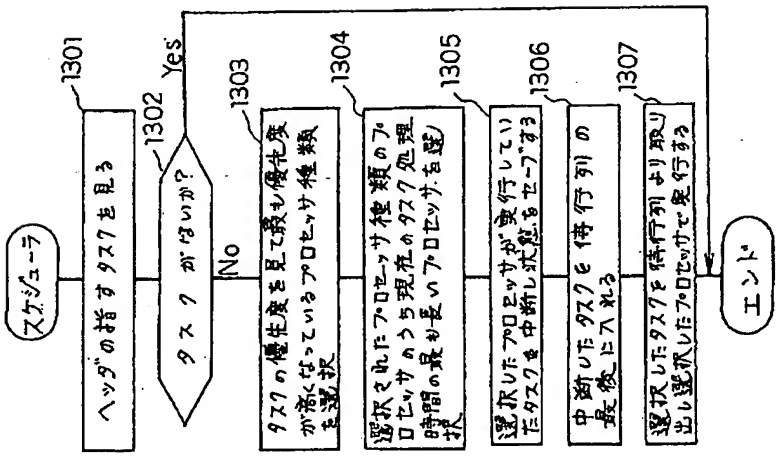
第 11 図



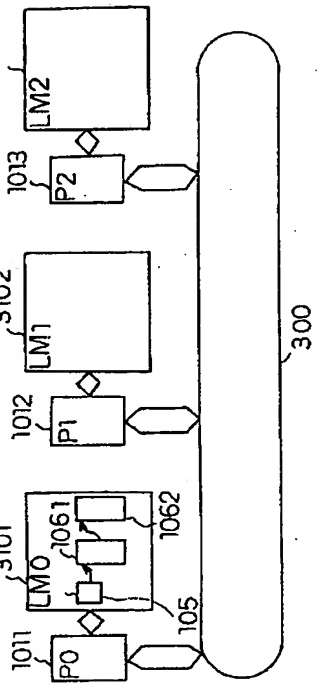
第 12 図



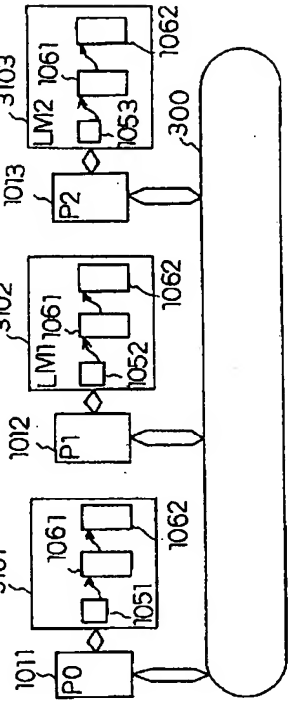
第 13 図



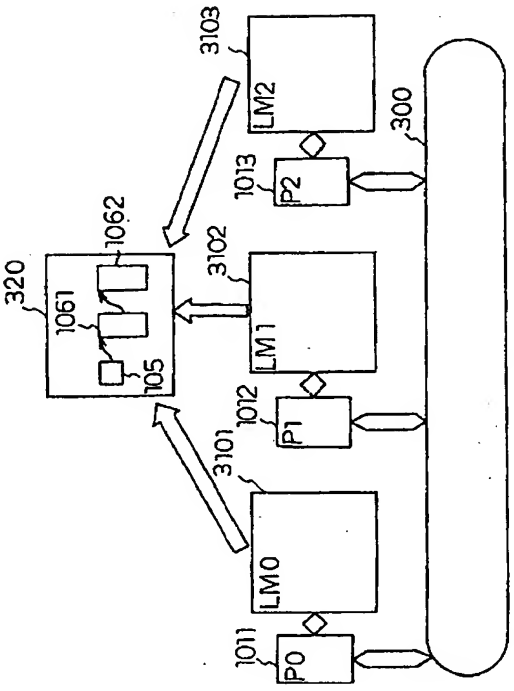
第 14 図



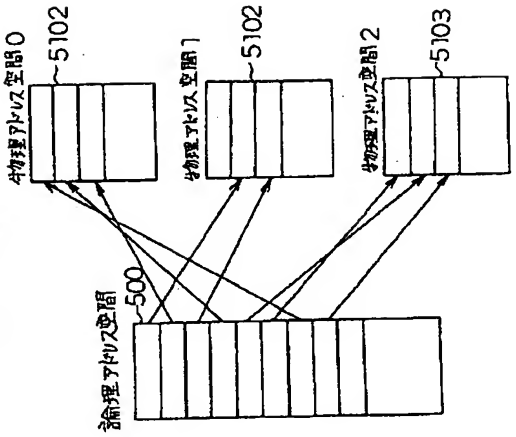
第 15 図



第 16 図



第 17 図



第 18 図

論理アドレス 0	プロセッサID 0	物理アドレス 0
〃 1	〃 1	〃 1
〃 2	〃 2	〃 2
〃 3	〃 3	〃 3
・・・	・・・	・・・
論理アドレス n	プロセッサID n	物理アドレス n

第1頁の続き

⑫発 明 者　中 村　　智 明　茨城県日立市大みか町5丁目2番1号　株式会社日立製作所大みか工場内